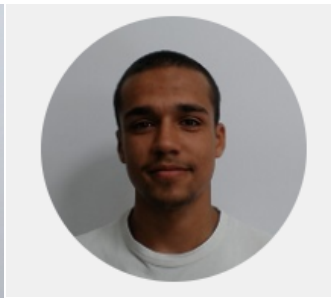
Projeto de Laboratórios de Informártica 3

Grupo 6

André Carvalho da Cunha Martins A89586
António Jorge Nande Rodrigues A89585
Rui Emanuel Gomes Vieira A89564


A89586


A89585


A89564

Resumo

Para a segunda parte do trabalho prático desta unidade curricular foi-nos proposto a implementação de um Sistema de Gestão de Vendas, SGV, mas desta vez na linguagem JAVA. Tal como no projeto em C foi-nos pedido para analisar três documentos de texto e que processámos essas informações de modo a responder a um conjunto de novas queries. Devido às ferramentas disponibilizadas pela linguagem e pela maior liberdade de trabalho com a mem ória o grupo considerou esta parte mais agradável de trabalhar.

1. Tipos e Estruturas de Dados

De forma a conseguir responder às queries de forma correta eficiente, após análise da informação contida nos ficheiros disponibilizados e dos requisitos de cada uma das queries, decidimos quais eram os métodos que iríamos utilizar para armazenar as informações.

1.1 Tipos de Dados

De forma a facilitar o funcionamento das queries distribuímos os dados pelas seguintes classes, sendo que vamos explicar a sua organização já de seguida.

1.2 Organização das estruturas elementares

De seguida vamos passar a falar sobre as classes que considerámos principais no armazenamento da informação.
O CatProd é a classe responsável pelo armazenamento dos dados lidos do ficheiro dos produtos. Optámos por guardar a informação num TreeSet de forma a não existirem elementos repetidos.
O CatClient destina-se a ser a classe responsável pelo armazenamento dos dados lidos do ficheiro dos clientes. Tal como na classe descrita anteriormente foi utilizado um TreeSet pelos mesmos motivos.
A classe Filial, distribui a informação lida no ficheiro das vendas por dois maps. Devido à maior eficácia na respota às queries optamos por optar por Treemaps. O primeiro map, responsabiliza-se por armazenar informação por cada filial. Deste modo, as chaves do map são o número de cada filial e os values são uma lista que contém as vendas respetivas à filial indicada por cada chave, 0 para vendas gerais. O segundo map, organiza as vendas por cliente, sendo cada chave o código de um cliente e os values uma lista de vendas de cada cliente.
A classe Faturação funciona de certo modo como a classe anterior mas o map que esta armazena é a relação entre os produtos e as vendas de cada um. Utilizando de novo um TreeMap, usámos como chave os códigos dos produtos e como values a lista de vendas de cada um. A classe MainStruct é a classe responsável por agregar as quatro classes supracitadas, pois as suas variáveis de instâncias são um CatProd, um CatClient, uma Faturação e uma Filial. Por fim, a última classe que consideramos principal no armazenamento da informação é a GestVendas, pois é aquela que é responsável por guardar a informação lida nas respetivas classes armazenadoras. Através da informação que é lida pela classe Loader, a classe Mainstruct e as suas v.i.'s são preenchidas, ficando a informação, deste modo, armazenada.

1.3 Outras classes auxiliares

Como classes auxiliares considerámos a Loader, Cliente, Product e VendasReader e as classes criadas para cada query.
As quatro primeiras classes auxiliares enunciadas são responsáveis pela leitura e armazenamento de alguma informação (Loader) dos dados dos ficheiros de texto. As classes Reader(Cliente, Produto e Vendas) são responsáveis por ler os ficheiros de texto. A classe Loader fica então responsável por associar a informação lida aos respetivos catálogos, faturação ou filial. Esta classe guarda também v.i's sobre o path do ficheiro lido, bem como uma série de resultados que são usados para fins estatísticos durante a execução do programa.

As classes criadas para as queries não têm outra utilidade para além de organização de código pois contam apenas com os métodos que resolvem a respetiva query e devolvem a collection ou dado pretendido como resposta.
Criámos também quatro classes de exceptions, responsáveis por prevenir uma ocorrência anormal que levasse ao mau funcionamento do programa. São então estas exceptions: NoClienteFoundException, que sinaliza quando é introduzido como input um cliente que não figura na base de dados; NoProductFoundException, que sinaliza quando é introduzido como input um produto que não pertence à base de dados; WrongBranchException, que deteta quando uma filial passada como input não corresponde a uma filial da empresa; WrongMonthException, responsável por verificar que não é passado um inteiro que não representa um mês.

2 Queries

Passamos assim à explicação da abordagem que utilizamos em relação a cada query.

Query 1 – Produtos nunca comprados e respetivo total

Para responder a esta query obtemos o catálogo de produtos e para cada um que pertence a esse set verificamos se esse código é uma chave do map das vendas por produto. Se não for, o produto é adicionado a um set que contém os produtos que nunca foram comprados. O método responsável por responder à query devolve um TreeSet pois foi o tipo de set que obteve um melhor desempenho.

Query 2 – Número total de vendas realizadas e número de clientes distintos que as fizeram(por filial)

O método que responde a esta query retorna um map cujas keys são códigos de clientes e os values são o número de compras que cada cliente fez.

Query 3 - Compras, produtos e total gasto de um dado cliente por mês

Para a resolução desta query criamos uma estrutura auxiliar que nos permite guardar informações(nº de compras, produtos e total gasto) respetivas a um determinado mês. Após filtrar as vendas por mês vamos preencher dar set as v.i.'s da classe auxiliar com os respetivos dados. O método retorna um arraylist

de classe auxiliar Q3MonthInfo com as informações relativas a cada mês de compras de um dado cliente.

Query 4 - Número de vendas, total faturado e clientes que compraram um dado produto

Tal como para a query anterior, para esta também é criada uma classe auxiliar que armazena a quantidade de vendas, um set de clientes que compraram o produto e o total faturado. A partir da lista de vendas obtida pela faturação vamos criar um map onde a key são os meses e os values são uma classe auxiliar Q4MonthInfo, com as informações relativas a esse produto no respetivo mês.

Query 5 - Produtos mais comprados por um dado cliente

O método vai retornar a lista das vendas de um cliente dado, organizada por um comparador de quantidade por nós criado.

Query 6 - X produtos mais vendidos em todo o ano e número de clientes que os compraram

Para responder a esta query usamos dois métodos. O primeiro vai retornar um set ordenado de um maps, onde as keys são códigos de um produto e os values a quantidade de clientes que os compraram. O segundo método retorna um map onde, novamente, a chave são os códigos dos produtos e os values são sets que contém os códigos dos clientes que compraram o produto que é a sua chave.

Query 7 - 3 maiores compradores de cada filial

Novamente para responder a esta query usámos dois métodos, onde o primeiro retorna um list de maps, cujas keys são códigos de clientes e os values são a total gasto por cada um. O segundo método, auxiliar, é o reponsável por criar e construir o map de cada filial, que posteriormente é organizado.

Query 8 - X clientes que compraram mais produtos diferentes

O método que responde a esta query retorna um set de uma classe auxiliar Pair, que conta com 2 v.i.'s, uma para guardar o código de um cliente e outra para guardar a quantidade de produtos diferentes que este comprou. O set de Pairs

está organizado por ordem decrescente de produtos diferentes comprados. No caso da quantidade ser a mesma, o set assume uma organização alfabética.

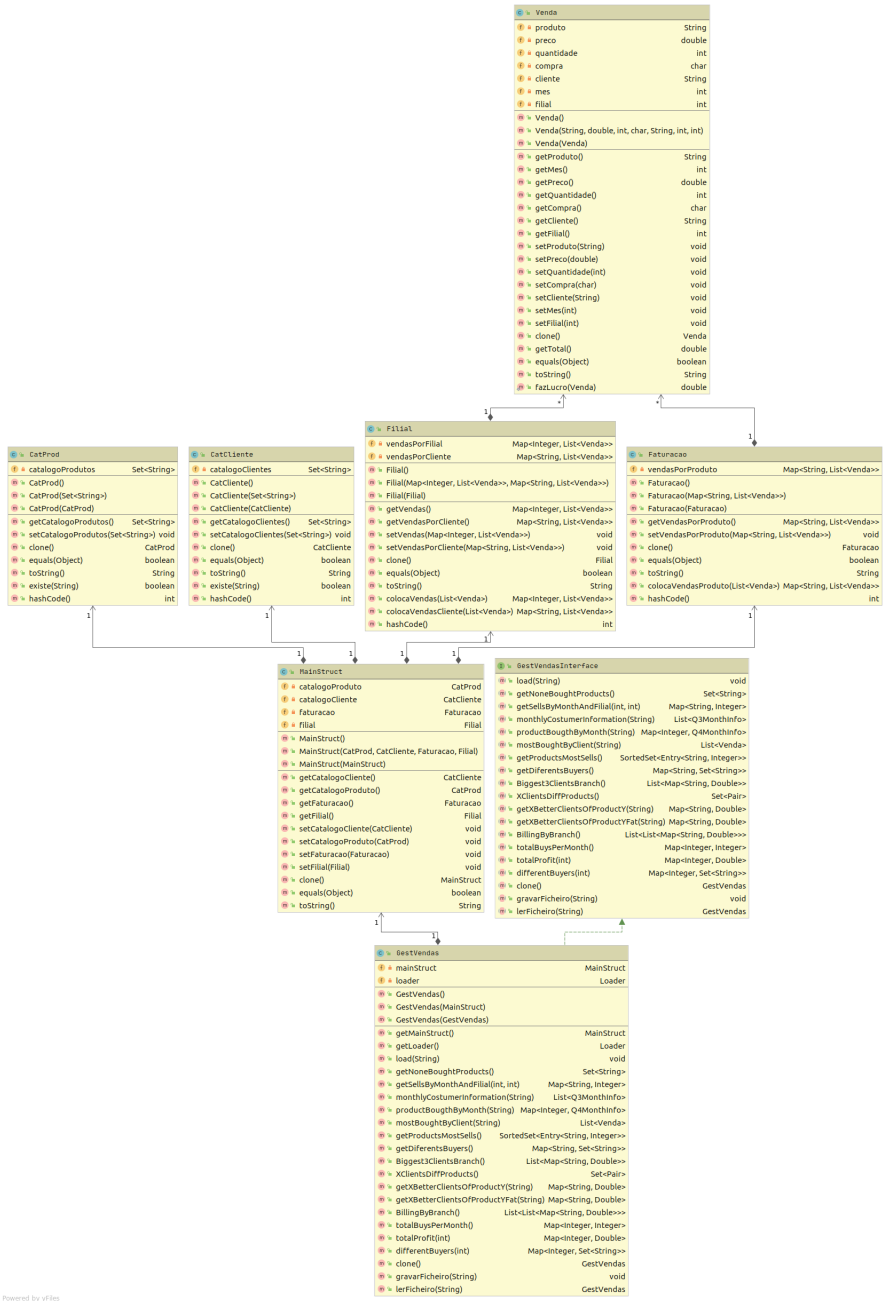
Query 9 - X clientes que compram um dado produto e valor gasto

Criámos dois métodos para responder a esta query. O primeiro que retorna um map, ordenado por ordem descrescente de values, onde as keys são os códigos dos clientes e os values são a quantidade comprada de um dado produto. O segundo método segue a mesma linha de pensamento mas os values, ao invés de serem a quantidade comprada, são o total gasto nesse produto por cliente.

Query 10 - Faturação total com cada produto, filial a filial, mês a mês

Criámos 3 métodos para conseguir responder a esta query. O principal que retorna uma lista cujo componentes são listas de maps. O método de menor nível é responsável por criar um map onde as keys são códigos de produtos e os values são o total gasto com cada produto num determinado mês, numa determinada filial. A função do nível médio é responsável por retornar uma lista que conta com um map para cada filial, respetivamente a um mês. A função principal agrupa na lista que retorna as listas correspondentes aos 12 meses do ano.

3 Grafo de dependências



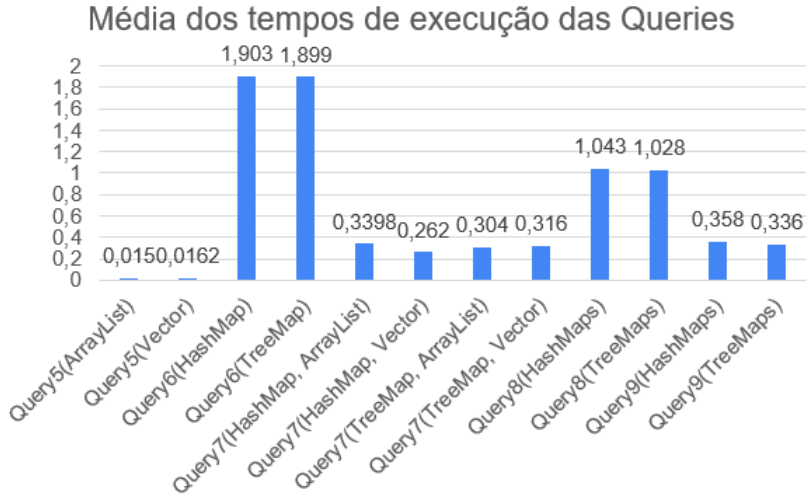
4 Testes de perfomance

Foi-nos também pedido que realizassemos alguns testes de performance e que fizessemos algumas medições de tempos, alterando estruturas de dados entre os testes. Dado os resultados apresentados, concluímos que usar estruturas do tipo HashMap ou HashSet se revela mais rápido do que o uso de TreeMaps ou TreeSet, ainda que não seja uma diferençã muito significativa. Contudo, existem situações em que é imperativo utilizar Trees, visto que é preciso manter uma ordenação. Em relação à comparação dos tempos de load usando a classe BufferedReader ou a classe Files revelam mais algumas diferenças, também pouco significativas, no tempo de load. Optamos por utilizar a classe BufferedReader, pois revela-se mais rapida quando era necessário carregar ficheiros maiores (3M e 5M). Estes testes serviram para nos mostrar a diferença entre o uso de várias estruturas e a importância de fazermos essas comparações para saber quais são as melhores opções e quais são as mais adequadas a serem utilizadas. As imagens sobre os dados sobre os testes de perfomance começam na página 7. Para os testes foi usado um computador processador 2,3 GHz Intel Core i5 de núcleo duplo, com memória 16 GB 2133 MHz LPDDR3 e sistema operativo macOS Catalina.

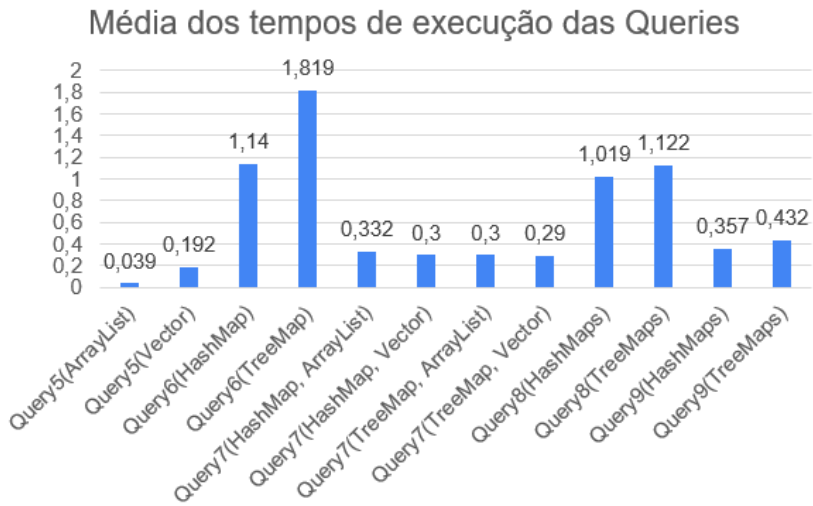
5 Conclusão

Tal como esperávamos, o desenvolvimento deste projeto foi faciltado usando Java, dado que, o leque de estruturas de dados nativas da linguagem é muita maior, quando comparado com C, e foi nos possível utilizar diversas estruturas de uma maneira mais simples e intuitiva. Em relação ao encapsulamento, este também é bastante mais fácil utilizando Java, visto que já é algo intrínseco da programação orientada a objetos (declaração de variáveis de instância como sendo private e posterior uso de gets, sets e clone, não havendo, assim, a partilha de apontadores. Se tívessemos que escolher uma linguagem para desenvolver este tipo de projeto, escolheríamos Java, devido ao número de opções de estruturas de dados que oferece e devido à maior facilidade de reutilizar o código, como por exemplo, para fazer uma mudança de TreeMap para um HashMap são precisas muitas poucas mudanças no código, o que facilita muito mais a reutilização do código, o que poupa imenso tempo ao programador.

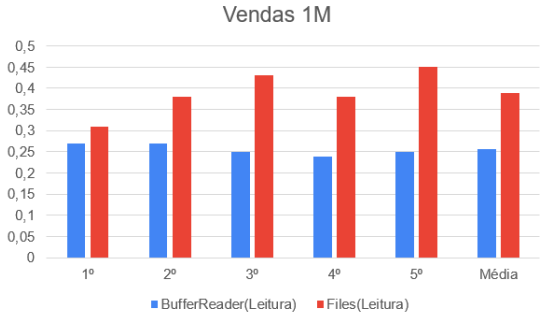
CatCliente	CatProd	Filial	Faturação								
TreeSet	TreeSet	TreeMaps	TreeMap								
	1º	2º	3º	4º	5º	6º	7º	8º	9º	10º	Média
Load	8.95	8.69	8.53	8.87	8.43	8.6	9.28	8.77	8.3	9.58	8,8
Query5(ArrayList)	0.02	0.014	0.013	0.014	0.011	0.012	0.014	0.016	0.019	0.017	0,015
Query5(Vector)	0.03	0.014	0.014	0.016	0.013	0.014	0.015	0.016	0.015	0.015	0,0162
Query6(HashMap)	1.7	1.9	2.2	1.7	1.6	2.12	1.76	1.88	1.97	2.2	1,903
Query6(TreeMap)	1.66	1.8	1.8	2.6	1.7	2	1.54	1.66	2.050	2.18	1,899
Query7(HashMap, ArrayList)	0.428	0.57	0.27	0.33	0.33	0.31	0.31	0.3	0.27	0.28	0,3398
Query7(HashMap, Vector)	0.38	0.26	0.25	0.27	0.23	0.25	0.25	0.24	0.26	0.23	0,262
Query7(TreeMap, ArrayList)	0.47	0.31	0.29	0.28	0.28	0.27	0.28	0.32	0.29	0.25	0,304
Query7(TreeMap, Vector)	0.46	0.31	0.31	0.32	0.28	0.28	0.3	0.31	0.28	0.31	0,316
Query8(HashMap, HashSet)	1.26	1.83	0.87	0.92	0.92	0.97	0.90	01.05	0.85	0.86	1.043
Query8(TreeMap, TreeSet)	0.88	1,02	0.95	0.89	1.050	0.88	0.91	1,16	1.68	0.86	1.028
Query9(HashMaps)	0.44	0.31	0.31	0.44	0.3	0.31	0.49	0.34	0.31	0.33	0,358
Query9(TreeMaps)	0.34	0.38	0.32	0.3	0.3	0.3	0.49	0.29	0.31	0.33	0,336

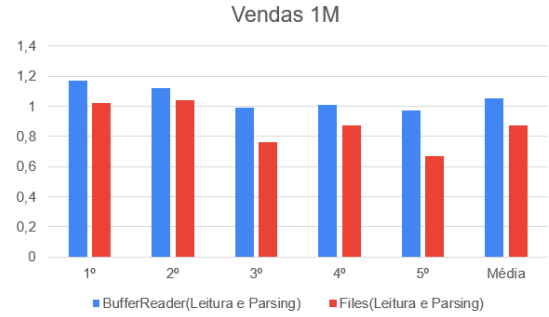


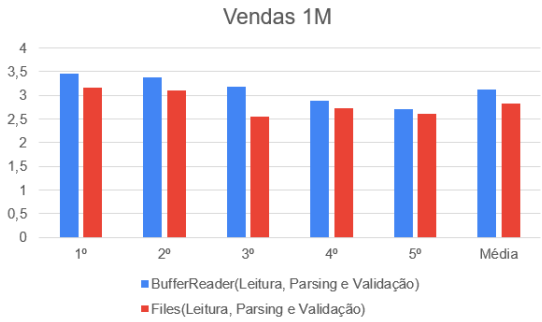
CatCliente	CatProd	Filial	Faturação								
HashSet	HashSet	HashMaps	HashMap								
	1º	2º	3º	4º	5º	6º	7º	8º	9º	10º	Média
Load	8.5	8.5	8.3	8.3	8.6	8.6	8.4	8.4	8.5	8.5	8,4
Query5(ArrayList)	0.03	0.02	0.018	0.023	0.0169	0.015	0.017	0.018	0.22	0.02	0,039
Query5(Vector)	0.027	0.036	0.017	0.015	0.017	0.015	0.015	0.015	0.02	0.015	0,192
Query6(HashMap)	1.7	2.051	2.34	2.032	2.014	2.043	1.77	1.9	1.9	2.3	1,14
Query6(TreeMap)	2.24	1.77	1.55	1.68	1.74	2.091	1.61	1.68	1.74	2.094	1,819
Query7(HashMap, ArrayList)	0.53	0.3	0.29	0.28	0.42	0.29	0.29	0.3	0.32	0.3	0,332
Query7(HashMap, Vector)	0.44	0.32	0.29	0.29	0.27	0.26	0.28	0.31	0.28	0.27	0,3
Query7(TreeMap, ArrayList)	0.4	0.34	0.32	0.28	0.28	0.31	0.3	0.28	0.28	0.28	0,3
Query7(TreeMap, Vector)	0.44	0.3	0.28	0.31	0.27	0.27	0.28	0.29	0.27	0.28	0,29
Query8(HashMap, HashSet)	1.18	1.1	0.92	0.91	1.23	1	1.080	0.95	0.85	0.97	1.019
Query8(TreeMap, TreeSet)	2.21	1.070	0.98	0.85	0.92	1	1.19	1.26	0.83	0.91	1.122
Query9(HashMaps)	0.38	0.31	0.31	0.3	0.5	0.3	0.31	0.31	0.31	0.54	0,357
Query9(TreeMaps)	0.34	0.63	0.43	0.31	0.33	0.42	0.31	0.64	0.31	0.6	0,432

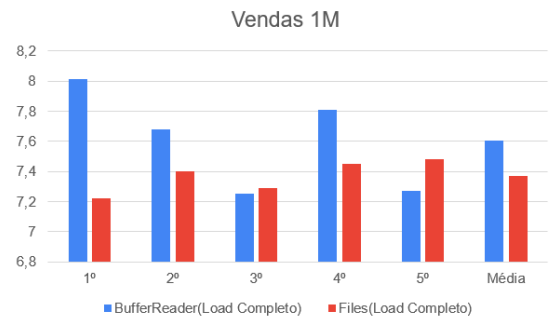


Vendas1M	1º	2º	3º	4º	5º	Média
BufferReader(Leitura)	0,27	0,27	0,25	0,24	0,25	0,256
BufferReader(Leitura e Parsing)	1,17	1,12	0,99	1,01	0,97	1,052
BufferReader(Leitura, Parsing e Validação)	3,47	3,39	3,18	2,89	2,71	3,128
BufferReader(Load Completo)	8,01	7,68	7,25	7,81	7,27	7,604
Files(Leitura)	0,31	0,38	0,43	0,38	0,45	0,39
Files(Leitura e Parsing)	1,02	1,04	0,76	0,87	0,67	0,872
Files(Leitura, Parsing e Validação)	3,16	3,11	2,56	2,72	2,62	2,834
Files(Load Completo)	7,22	7,4	7,29	7,45	7,48	7,368

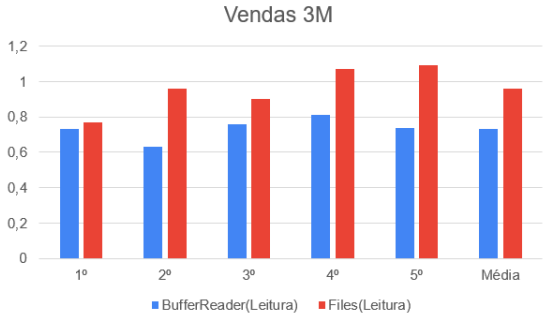


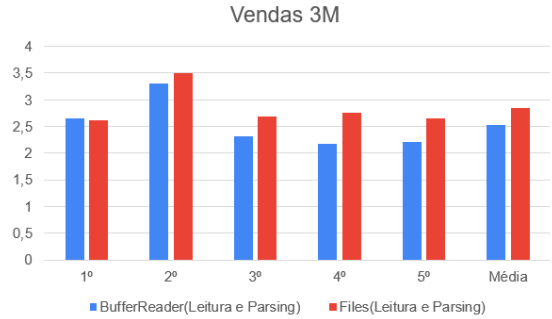


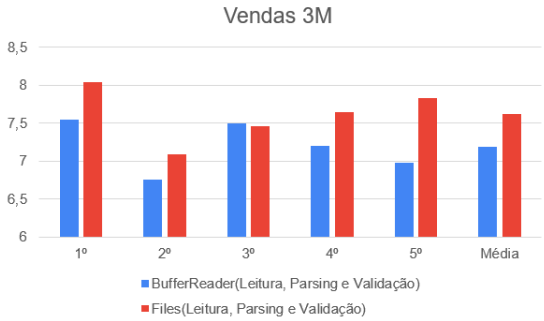


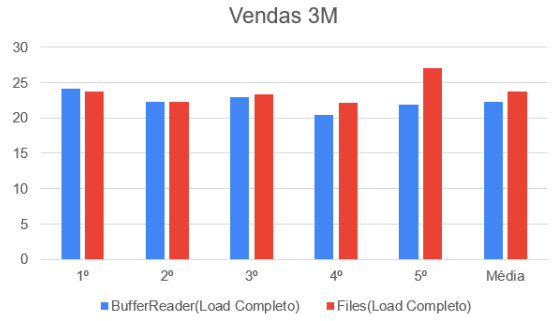


Vendas3M	1º	2º	3º	4º	5º	Média
BufferReader(Leitura)	0,73	0,63	0,76	0,81	0,74	0,734
BufferReader(Leitura e Parsing)	2,65	3,3	2,31	2,17	2,21	2,528
BufferReader(Leitura, Parsing e Validação)	7,54	6,75	7,49	7,2	6,98	7,192
BufferReader(Load Completo)	24,11	22,25	22,93	20,49	21,9	22,336
Files(Leitura)	0,77	0,96	0,9	1,07	1,09	0,958
Files(Leitura e Parsing)	2,61	3,5	2,68	2,76	2,65	2,84
Files(Leitura, Parsing e Validação)	8,04	7,09	7,46	7,65	7,83	7,614
Files(Load Completo)	23,77	22,27	23,4	22,11	27,04	23,718

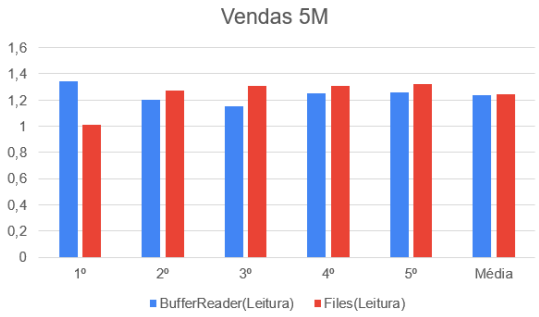


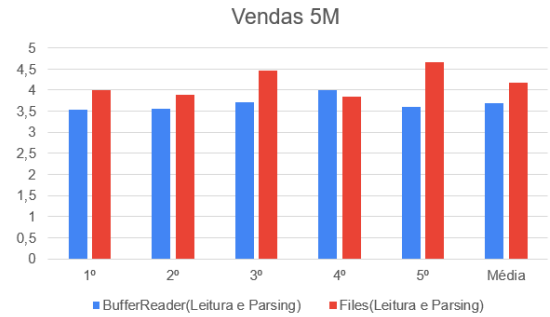


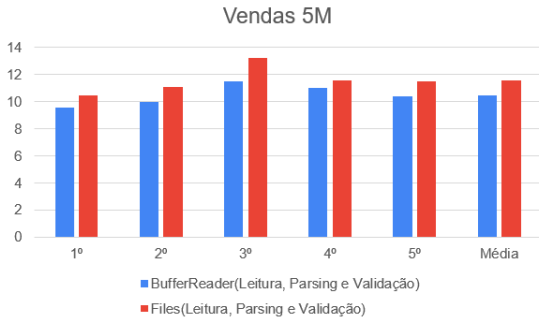


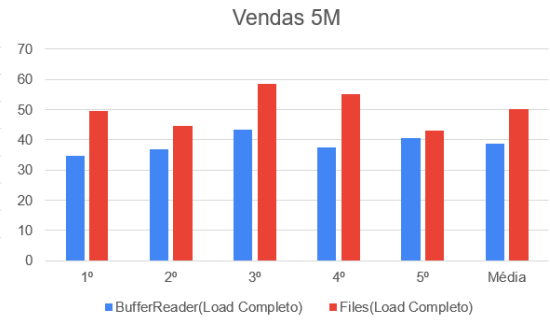


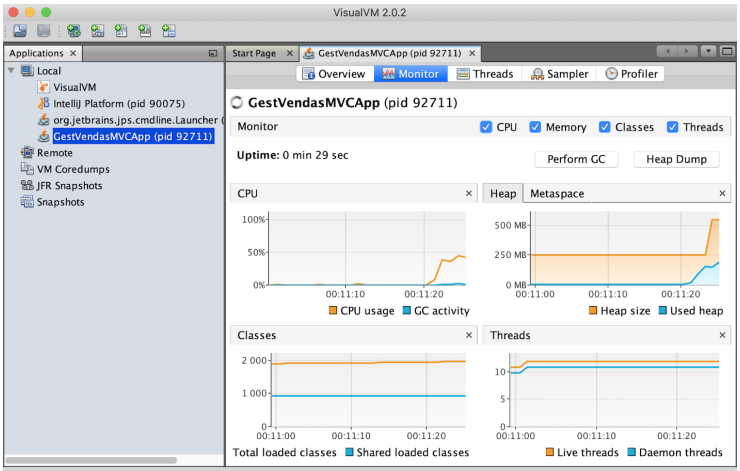
Vendas5M	1º	2º	3º	4º	5º	Média
BufferReader(Leitura)	1,34	1,2	1,15	1,25	1,26	1,24
BufferReader(Leitura e Parsing)	3,53	3,55	3,71	4	3,61	3,68
BufferReader(Leitura, Parsing e Validação)	9,58	9,95	11,46	11	10,37	10,472
BufferReader(Load Completo)	34,8	36,75	43,35	37,49	40,6	38,598
Files(Leitura)	1,01	1,27	1,31	1,31	1,32	1,244
Files(Leitura e Parsing)	4	3,89	4,46	3,85	4,66	4,172
Files(Leitura, Parsing e Validação)	10,48	11,05	13,25	11,54	11,48	11,56
Files(Load Completo)	49,58	44,63	58,52	55,21	42,92	50,172

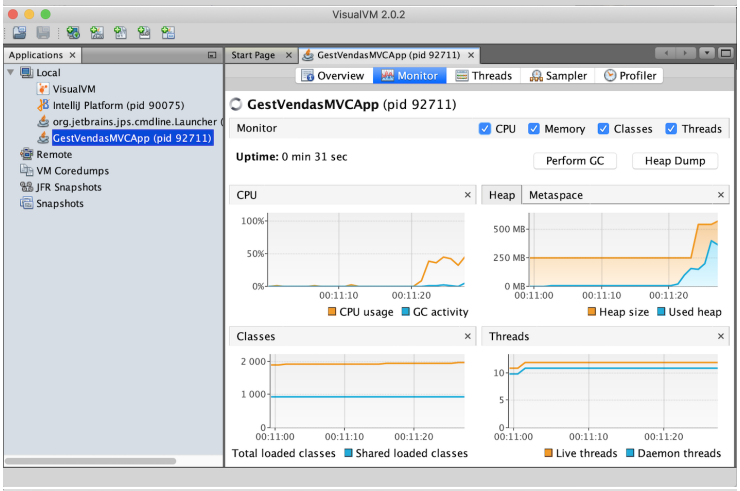


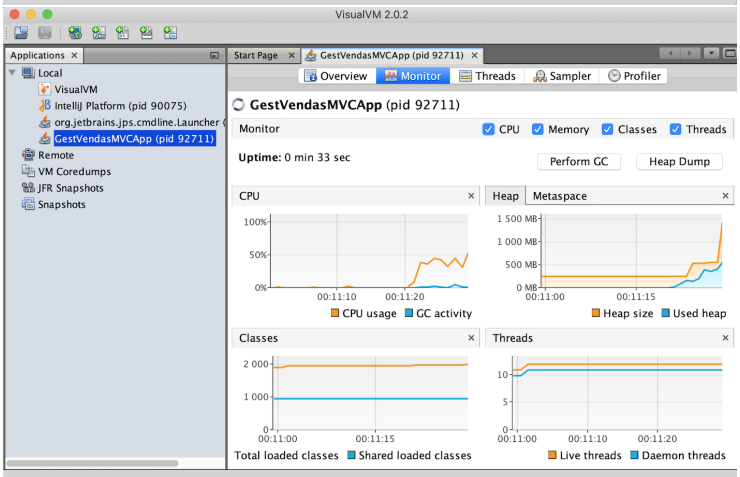


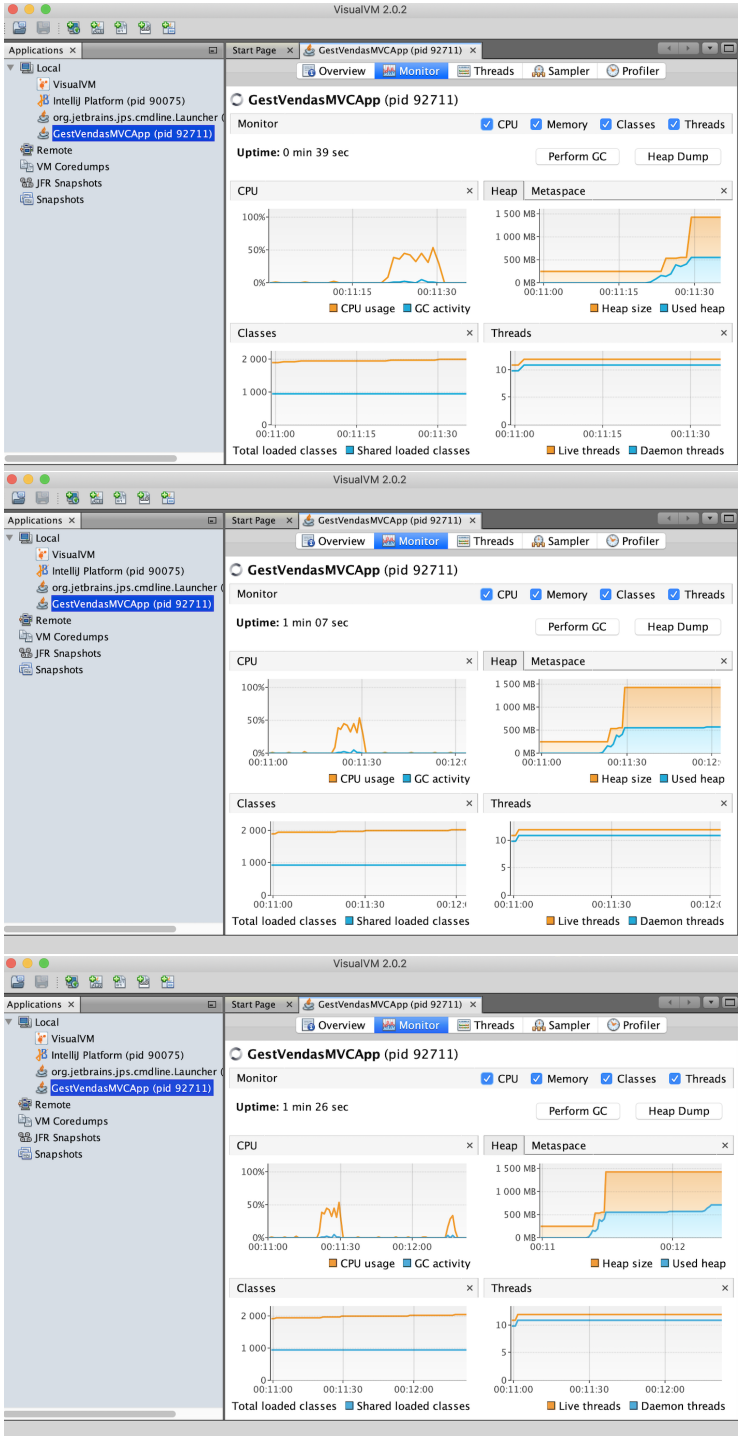


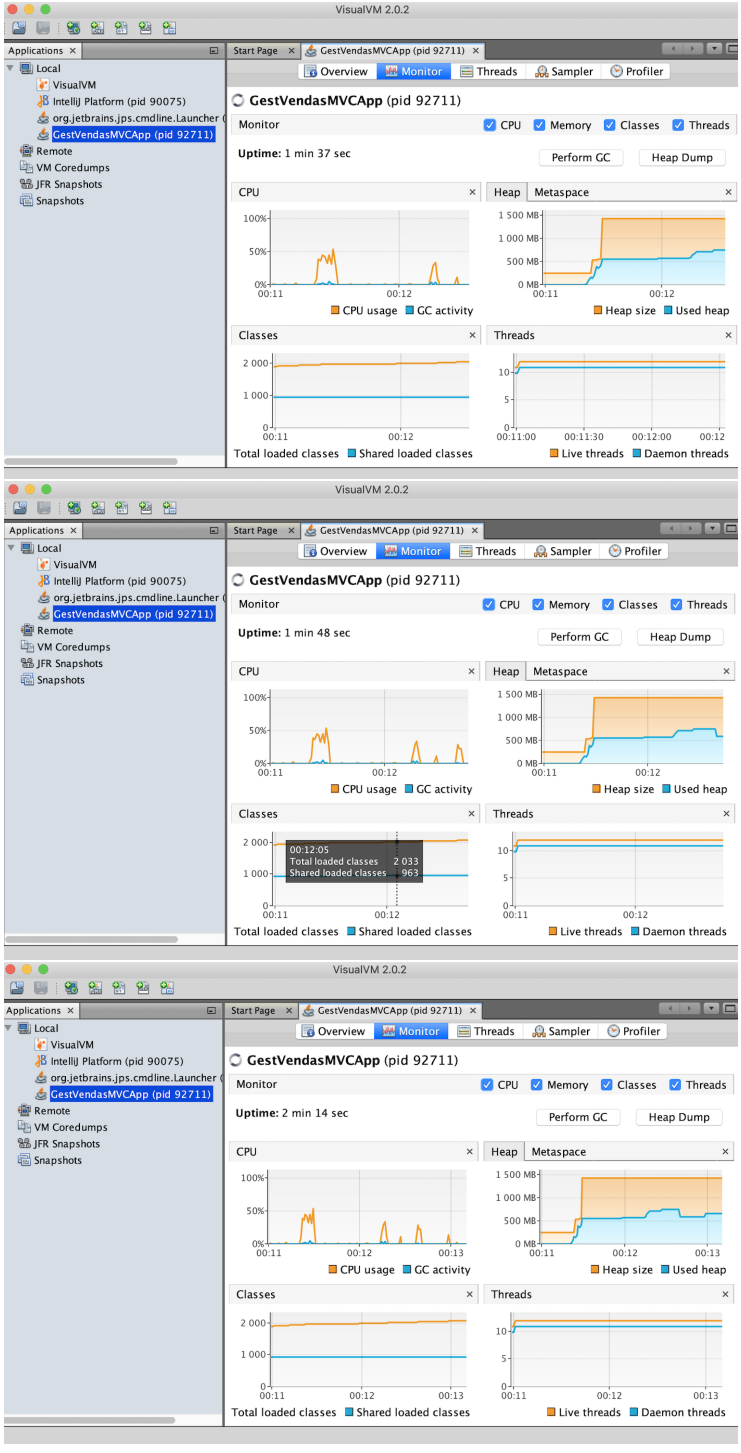



VisualVM 2.0.2
Applications
Local
VisualVM
Intellij Platform (pid 90075)
org.jetbrains.jps.cmdline.Launcher
GestVendasMVCApp (pid 92711)
Remote
VM Coredumps
JFR Snapshots
Snapshots
Start Page
GestVendasMVCApp (pid 92711)
Overview
Monitor
Threads
Sampler
Profiler
GestVendasMVCApp (pid 92711)
Monitor
CPU
Memory
Classes
Threads
Uptime: 0 min 29 sec
Perform GC
Heap Dump
CPU
Heap
Metaspace
100%
50%
0%
00:11:10
00:11:20
CPU usage
GC activity
500 MB
250 MB
0 MB
00:11:00
00:11:10
00:11:20
Heap size
Used heap
Classes
2 000
1 000
0
00:11:00
00:11:10
00:11:20
Total loaded classes
Shared loaded classes
Threads
10
5
0
00:11:00
00:11:10
00:11:20
Live threads
Daemon threads


VisualVM 2.0.2
Applications
Local
VisualVM
Intellij Platform (pid 90075)
org.jetbrains.jps.cmdline.Launcher
GestVendasMVCApp (pid 92711)
Remote
VM Coredumps
JFR Snapshots
Snapshots
Start Page
GestVendasMVCApp (pid 92711)
Overview
Monitor
Threads
Sampler
Profiler
GestVendasMVCApp (pid 92711)
Monitor
CPU
Memory
Classes
Threads
Uptime: 0 min 31 sec
Perform GC
Heap Dump
CPU
Heap
Metaspace
100%
50%
0%
00:11:10
00:11:20
CPU usage
GC activity
500 MB
250 MB
0 MB
00:11:00
00:11:10
00:11:20
Heap size
Used heap
Classes
2 000
1 000
0
00:11:00
00:11:10
00:11:20
Total loaded classes
Shared loaded classes
Threads
10
5
0
00:11:00
00:11:10
00:11:20
Live threads
Daemon threads


VisualVM 2.0.2
Applications
Local
VisualVM
Intellij Platform (pid 90075)
org.jetbrains.jps.cmdline.Launcher
GestVendasMVCApp (pid 92711)
Remote
VM Coredumps
JFR Snapshots
Snapshots
Start Page
GestVendasMVCApp (pid 92711)
Overview
Monitor
Threads
Sampler
Profiler
GestVendasMVCApp (pid 92711)
Monitor
CPU
Memory
Classes
Threads
Uptime: 0 min 33 sec
Perform GC
Heap Dump
CPU
Heap
Metaspace
100%
50%
0%
00:11:10
00:11:20
CPU usage
GC activity
1 500 MB
1 000 MB
500 MB
0 MB
00:11:00
00:11:15
Heap size
Used heap
Classes
2 000
1 000
0
00:11:00
00:11:15
Total loaded classes
Shared loaded classes
Threads
10
5
0
00:11:00
00:11:10
00:11:20
Live threads
Daemon threads


VisualVM 2.0.2
Applications
Local
VisualVM
Intellij Platform (pid 90075)
org.jetbrains.jps.cmdline.Launcher
GestVendasMVCApp (pid 92711)
Remote
VM Coredumps
JFR Snapshots
Snapshots
Start Page
GestVendasMVCApp (pid 92711)
Overview
Monitor
Threads
Sampler
Profiler
GestVendasMVCApp (pid 92711)
Monitor
CPU
Memory
Classes
Threads
Uptime: 0 min 39 sec
Perform GC
Heap Dump
CPU
Heap
Metaspace
100%
50%
0%
00:11:15
00:11:30
CPU usage
GC activity
1 500 MB
1 000 MB
500 MB
0 MB
00:11:00
00:11:15
00:11:30
Heap size
Used heap
Classes
2 000
1 000
0
Total loaded classes
Shared loaded classes
Threads
10
5
Live threads
Daemon threads
VisualVM 2.0.2
Uptime: 1 min 07 sec
VisualVM 2.0.2
Uptime: 1 min 26 sec


VisualVM 2.0.2
Applications
Local
VisualVM
IntelliJ Platform (pid 90075)
org.jetbrains.jps.cmdline.Launcher
GestVendasMVCApp (pid 92711)
Remote
VM Coredumps
JFR Snapshots
Snapshots
Start Page
GestVendasMVCApp (pid 92711)
Overview
Monitor
Threads
Sampler
Profiler
GestVendasMVCApp (pid 92711)
Monitor
CPU
Memory
Classes
Threads
Uptime: 1 min 37 sec
Perform GC
Heap Dump
CPU
Heap
Metaspace
CPU usage
GC activity
Heap size
Used heap
Classes
Threads
Total loaded classes
Shared loaded classes
Live threads
Daemon threads
VisualVM 2.0.2
Uptime: 1 min 48 sec
00:12:05
Total loaded classes 2 033
Shared loaded classes 963
VisualVM 2.0.2
Uptime: 2 min 14 sec